Future Generation Computer Systems 96 (2019) 563–578

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Towards interactive networking: Runtime message inference approach for incompatible protocol updates in IoT environments☆,☆☆

Heesuk Son, Dongman Lee *

KAIST, 291, Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea

HIGHLIGHTS

- SeM2Bit infers an updated protocol message efficiently based on protocol knowledge.
- Build a message inference knowledge base by a thorough analysis on famous IoT protocols.
- Develop tree-based algorithms to compute a legacy message' adaptation sequences.
- Develop pruning mechanisms to help reduce the trees' search spaces and overhead.
- Conduct case study experiments and prove that SeM2Bit infers updated messages efficiently with low overhead.

ARTICLE INFO

Article history:
Received 14 August 2018
Received in revised form 20 December 2018
Accepted 5 February 2019
Available online 13 February 2019

Keywords:
Protocol adaptation
Interoperability
Protocol heterogeneity
Meaningful interaction
Internet-of-things

ABSTRACT

As IoT (Internet of Things) devices become pervasive in our surroundings, it is more important for them to dynamically discover and interact with nearby IoT devices. However, as interaction protocols are often updated without backward compatibility, interaction opportunities between smart objects may disappear. To overcome this, we can apply existing works which try to adapt one interaction protocol to another at runtime, assuming their specifications are given. However, they cannot generate a valid adapter if the specification of the updated protocol syntax is not available. Automatic protocol reverse engineering methods can extract protocol message syntax without prior knowledge, but they cannot be applied to a runtime scenario. In this paper, we propose SeM2Bit, an efficient protocol message inference scheme which adapts a legacy protocol's message based on protocol domain knowledge. Iterative message adaptations make a legacy protocol agent interact with its updated but incompatible version at runtime and have a meaningful interaction without the corresponding specification. The experiment result shows that the use of knowledge is effective to make a meaningful interaction between the incompatible versions with a reasonably small number of message adaptations.

1. Introduction

As Internet of Things (IoT) devices and their computing capabilities become pervasive in our surroundings, new computing environments such as edge environment are receiving a lot of attention. To satisfy diverse requirements from the new environments, novel architectures have been presented with respect to 5G [1], vehicular communications [2], network virtualization [3], etc. In such environments, users encounter more chances to interact with nearby devices, where both parties usually assume to use the same version of an interaction protocol. However, high-level requirement changes often induce the protocol updates [4,5], which hinders smart objects from proactively collaborating with each other. Especially, updates on a protocol message structure without backward compatibility disable even a simple interaction such as a request-reply between different versions. Suppose that a user brings a mobile phone running a Service Location Protocol version 1 (SLPv1) [6] agent for discovering a nearby IoT devices running SLP version 2 (SLPv2) [7] which has an incompatible message structure. Even though SLPv2 is available online, replacing SLPv1 with it is time-consuming and not feasible at runtime. To resolve such backward incompatible updates, we can apply interface adaptation schemes [8–11] which identify syntactic mismatches between the interfaces of two target protocols and resolve them based on similarity-based clustering. However, these schemes do not provide valid adaptation to protocol updates if the specification of an updated message syntax is not available just-in-time.

To derive an unknown message syntax of an updated protocol without prior knowledge, Automatic Protocol Reverse Engineering (APRE) tools can also be applied. From captured protocol

☆ Declarations of interest: none.

☆☆ This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01126, Self-learning based Autonomic IoT Edge Computing).

* Corresponding author.
E-mail addresses: heesuk.son@kaist.ac.kr (H. Son), dlee@kaist.ac.kr (D. Lee).

https://doi.org/10.1016/j.future.2019.02.007
0167-739X/

message traces [12,13], keyword candidates and their attributes (e.g. keyword transition probability, field position, etc.) are computed and a target message syntax is extracted. Executable-based APRE schemes [14,15] leverage a target protocol's executable for extracting message fields and their sequences from output buffers. However, it is hard to collect enough network traces generated by a specific protocol agent before an interaction opportunity disappear, especially, in a mobile environment. In addition, required efforts and resources for network monitoring, packet aggregations, and structure learning also lower their applicability. Moreover, acquiring a target protocol's executable is not feasible in most IoT environments.

An alternative for successful message exchanges is to infer a target protocol's message syntax. The most primitive inference method is a random brute force guess. However, too many possibilities make it hard to enable successful message exchanges in a reasonable time. As an effective alternative, domain knowledge can be exploited. In the past, LINCOS [16] tried to deliver a meaningful message by means of radio waves to "humanlike" recipients which do not share the common language. Its proposal stands on an insight that communicating parties may eventually understand the delivered meaning if messages are composed based on the laws of shared physical world. Juba [17], inspired by LINCOS, devised a knowledge-based efficient computation model of meaningful communications between two agents using different languages. He considers an interaction meaningful when a sender's statement is translated into the same statement in the receiver's language and they can achieve an effect through the interaction. However, Juba fails to realize his scheme to a real world protocol update scenario with a concrete architecture and its time complexity is too high.

For verification purpose, Juba assumes an oracle that judges if a message from a sender and its translation by a receiver are identical. However, since it is not always possible to assume the existence of the oracle, we refine the definition of a meaningful interaction and its verification as follows: a sender's intention is understood by the receiver and a non-error reply message is returned as expected. To enable a meaningful interaction with an updated protocol before interaction opportunities are gone, it is necessary to construct a basis to make an inference quickly on probable updates with. An inference becomes more difficult since not a single message field but a sequence of message field updates should be inferred correctly in a right order as a protocol update entails. Since the possibilities rise drastically, it is required to narrow down the search space to reduce the complexity of the valid sequence detection.

In this paper, we propose an efficient protocol message inference scheme for meaningful interactions between incompatible protocol versions, called SeM2Bit. By leveraging protocol domain knowledge, SeM2Bit adapts a legacy message to its updated version at runtime without corresponding specifications, protocol message samples, and a protocol executable. For this, the proposed scheme features three key components: (1) a knowledge base that abstracts the message structures and update patterns of existing protocols, (2) tree-based algorithms to identify adaptation method candidates for updating a legacy message to its update and compute their sequences, and (3) pruning mechanisms which reduce the number of possible adaptation methods by removing meaningless future candidates from the trees. For knowledge construction, we compare the characteristics of the existing protocols and extract their commonalities and differences. Then, the knowledge is transformed into a weighted tree to tell the most likely features to be added or removed. Eventually, the tree tells how to adapt a legacy protocol message that a user agent currently knows of to an updated one. The adapted message is transmitted to the counterpart and this adaptation-transmission routine repeats until a meaningful interaction is made. This top-down approach (Semantic-to-Bit level) helps to avoid random bit-level message updates and efficiently narrow down the adaptation field search space.

For evaluation, we conduct two case studies with two representative interaction protocols in IoT environments: Service Location Protocol (SLP) and Message Queuing Telemetry Transport (MQTT). In SLP case study, an IoT device uses SLPv1 and tries to interact with an encountered IoT device using SLPv2. In MQTT case study, an IoT device using MQTT version 3.1.1 tries to connect to a MQTT broker conforming with MQTT version 5. We implement SeM2Bit in Java on raspberry pi 3 Model B boards as IoT devices. The SLP case study results show that SeM2Bit enables meaningful interactions despite 0.7 of adaptation accuracy and 0.25 of semantic loss in average. In the MQTT case, SeM2Bit enables meaningful interactions despite 0.27 of precision degradation, but no semantic loss. Both results show that exploiting the protocol knowledge base is effective to achieve meaningful interactions in a reasonable elapsed time. In terms of memory overhead, SeM2Bit incurs 10.98 MB and 10.81 MB in the best case and up to 27.24 MB and 12.92 MB in the worse case, respectively in SLP and MQTT case.

In summary, our contributions are as follows:

- We study the incompatible protocol update problem that often happens and disrupts pervasive interactions in IoT environments. To achieve meaningful interactions between incompatible protocol versions, we propose an efficient approach to infer the updated message without corresponding protocol specifications at runtime.
- For the runtime message inference, we devise three key mechanisms. First, we construct an ontology-based protocol knowledge via a thorough analysis on widely-used IoT protocols, which gives a bias to infer possible protocol updates better. Then, we develop two tree-based algorithms which leverage the bias to compute probabilities of message fields' adaptation candidates and extract probable sequences of the candidates. To reduce the trees' extremely large search space and improve its efficiency, we develop pruning mechanisms which remove meaningless adaptation candidates.
- We prove that the proposed scheme infers updated messages efficiently with low overhead by narrowing down possible message update candidates through two case studies with the most famous IoT interaction protocols, SLP and MQTT.

The remainder of this paper is organized as follows. Section 2 describes existing works and their limitations in detail. Section 3 presents our design goal and describes key requirements for the knowledge-based efficient inference. Section 4 presents key components to overcome the existing works' limitations and accomplish an efficient inference. Section 5 explains evaluation results from two case studies. Section 6 discusses possible extensions of the current SeM2Bit. Finally, Section 7 provides a summary of our work and future plans.

2. Related works

2.1. Runtime interface adapter generation

In runtime interface-adaptation schemes [10,11], interaction protocols are represented by a set of interface syntax and a state machine which describes the behavior of the interfaces. Once the representations of two interacting parties are shared, they identify syntactic and behavioral mismatches which disrupt their interactions. Then, the syntactic mismatches are resolved based on interface similarity comparison techniques. If the behavior of the target protocol is not known, machine learning techniques such as active automata learning, L^* algorithm [18], are applied to learn the target protocol's messaging (or invocation) order. L^*

algorithm helps to setup a hypothetical behavior model of the target protocol agent and update the model by querying the agent repetitively. Eventually, two protocols' interaction model is generated by combining their state machines. This combined model is compiled and leveraged as an interface adapter at runtime. However, these approaches cannot generate a valid adapter to an updated protocol if the renewed specification of updated message syntax is not available just-in-time, which often happens in real life situations. In addition, the L* algorithm requires target protocol agents to react to every single request/query message by returning a counterexample, but in reality, incompatible agents usually drop the request message and do not reply. Therefore, it is not suitable to apply the L* algorithm for handling runtime protocol updates.

FP7 CONNECT project [8,9,19] presents *Emerging middleware* which generates an interface adapter at runtime. Emerging middleware listens to all the standard service discovery protocol ports (e.g. 1900 port for UPnP, 5353 port for mDNS, etc.) and captures their multicast advertisement/lookup messages. When a message is captured, it interacts with the message sender to extract its service interface list and generates a dedicated adapter at runtime. It resolves the syntactic mismatches by means of a machine learning-based interface similarity comparison technique. Then, it applies the L* algorithm [18] to learn with which behavior the given interfaces of two parties can achieve a successful interaction. As the result, a combined automata is produced as a runtime adapter. However, they also assume the full interoperability support in protocol message level. Since IoT devices do not bring all kinds of service discovery protocols with all different versions together, Emerging middleware cannot initiate its heterogeneity resolution process if it does not have a renewed specification or a binary code of an updated protocol message.

2.2. Automatic protocol reverse engineering

Automatic Protocol Reverse Engineering (APRE) tools can be leveraged to derive an updated protocol's message syntax without prior knowledge. Network trace-based approaches [12,13] capture message samples using packet capturing tools (e.g. WireShark) and infer a message syntax in terms of keyword position. From the collected messages, Luo [13] first detects frequent strings with small variance of positions as keyword candidates. Apriori algorithm [20] is used to mine the frequently associated strings. A sequence of zeros and ones composes a string in binary protocols (e.g. DNS), while a sequence of alphabets does in text-based protocols (e.g. HTTP). Then, Apriori algorithm is used again to compute frequent series of the identified strings as keyword sequences. Based on the extracted keyword sequence patterns, a given message is classified to one of the trained protocols. Biprominer [12] analyzes frequencies of each byte in binary protocol message traces and identifies blocks of bytes which appear together with a high probability. Then, Biprominer computes transition probabilities among the identified byte blocks. By following through the most likely sequence of byte blocks, a protocol message with consecutive keywords is estimated. Despite their clarity and great performance, collecting enough network traces transmitted by a target IoT object takes much more time than durations of valid interaction opportunities.

An updated protocol's executable can be also leveraged [14,15] for extracting its message structure. Programs usually store message fields in memory buffers in an inversed order and combine them together to compose a message to be sent. Hence, by deconstructing the memory buffer, they build a *message field tree*, a tree in which nodes represent message fields. Each node contains field attributes such as locations in a message. Then, by keeping track of how each field is used by program functions or instructions, semantics of each field are inferred. For example, when a field is used to derive an argument of a known function, its semantics can be easily inferred. The scheme is quite intuitive and widely used, especially in botnet protocol analysis domain. However, acquiring an executable of an encountered program is not feasible in IoT environments, especially at runtime. In case of [14], they use a custom environment which produces a list of executed instructions and the associated taint information to obtain program execution traces.

3. Design considerations

To achieve a meaningful interaction without the corresponding specification at runtime, we should be able to narrow down the possibilities based on domain knowledge. In this section, we clarify our design goal and describe key requirements for the knowledge-based efficient inference.

3.1. Definition of meaningful interactions

In Juba's work [17], an interaction is considered meaningful when (1) the sender's statement is translated into the same statement in the receiver's language and (2) they can achieve an effect through the interaction. However, in many interaction cases between IoT protocol agents, an intended effect can be achieved even without the translation into the same statement in the updated version of protocol in the receiver's side. Generally, in a request-reply interaction pattern, if the receiver can parse a request message, it sends a reply message back to the request sender. If a reply message brings a requested information or resource, the requester and receiver can achieve an intended effect (e.g. service lookup and notify) through the interaction. However, a returned reply message does not necessarily imply that the syntax and semantics of a request message inferred by a sender are as precise as expected by the receiver's updated protocol (i.e. *the same statement in the receiver's language* in Juba's definition). For example, it often happens that a field in a request message has a wrong value but that field is not used and ignored by the receiver (i.e. updated protocol's agent). Then, the request message is parsed without a problem and the receiver understands the core of the message's necessary operation semantics.

In this work, considering this circumstance, we relax Juba's definition of meaningful interaction and make it achievable in real situations. We define a meaningful interaction as an interaction where the receiver (1) understands the intention of a request message and (2) returns a non-error message. We hold this definition and aim to infer an updated protocol message to enable a meaningful interaction between incompatible versions as our design goal.

3.2. Knowledge base construction

A protocol update is usually initiated when new requirements are made as a consensus. Therefore, if we understand the likelihood of requirement changes and corresponding protocol updates, we can guess how differently the new version of a target protocol may operate and how to adapt the protocol's current behavior to the changes. However, since this generic understanding accommodates common characteristics across various protocols, it also includes irrelevant knowledge to an adaptation from a specific legacy protocol to its updated version. If invalid adaptations affect an existing legacy protocol repetitively and waste much of time, interaction opportunities of a mobile user will disappear due to meaningless message exchanges. Therefore, it is necessary to intelligently figure out a right set of probable requirement changes and adaptation method candidates so that random selections are prevented and then legitimate interactions continue to be achieved.

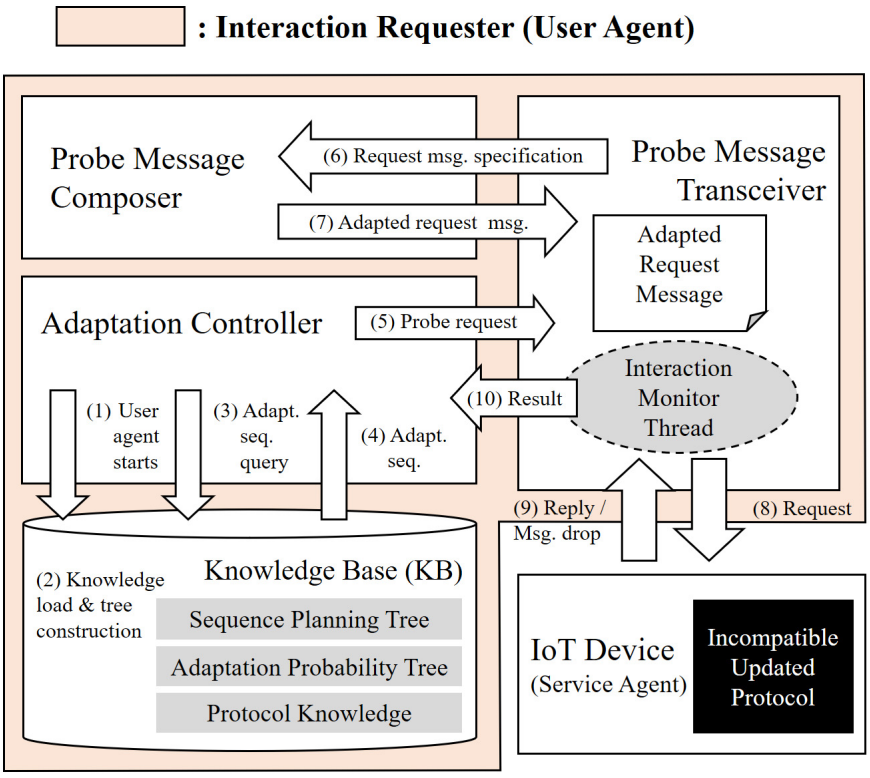


Fig. 1. A system architecture overview.

3.3. Valid adaptation sequence search

A protocol syntax update usually entails a sequence of various updates across different message fields such as deletion, addition, length change, and so on. In other words, it is not enough to find one valid message field adaptation to interact with a different version of a target protocol. Instead, a set of valid adaptations should be found and applied to update the legacy version of a target protocol message. A challenge is that an order of valid adaptations also matters. If the order is not correct, one adaptation (e.g. addition and deletion) on a message field makes other fields start at wrong positions, which will result in a receiver's failure to parse a message modified by such adaptation. Therefore, it is important to design a mechanism for finding out a right sequence of valid adaptation methods to avoid meaningless message exchanges and keep interaction opportunities valid.

3.4. Search space reduction

As our goal is to infer a right sequence of valid adaptations, not a single one, the number of possibilities drastically increases. For example, if 10 adaptation method candidates are identified, the number of possible sequences is almost 10 million and it is impossible to determine a right sequence in time. Furthermore, bringing all this information stored in memory and looking up the large search space require much computing resource overhead. The more protocols we target, the more resource overhead is required. Therefore, to achieve runtime interoperability in a real world scenario which include resource-constrained devices as well, it is critical to reduce a search space as much as possible.

4. Proposed scheme

In this section, we first describe a theoretical background which inspires the proposed scheme and list up its limitations to inferencing an updated protocol's message in a real situation. Then, we present key components to overcome the limitations and accomplish an efficient knowledge-based inference.

4.1. Theoretical background

Juba [21] formulates a computational model of communications between two different languages and suggests an iterative message modification scheme to enable the communication, which Juba calls *Universal Semantic Communication*. In this framework, a message sender leverages an interactive Turing machine to encode an output message of its streaming protocol. The sender keeps modifying output messages and sending them until the receiver correctly decodes these message in its own language. To optimize the modifications and establish an upper bound to the number of iterations, Exp3 algorithm [22] based on a domain knowledge is applied.

Despite its universality, Juba does not provide any architectural view of how to use domain knowledge for the Turing machine construction and how to incorporate Exp3 algorithm into it. In addition, Exp3 algorithm is not a proper choice for protocol adaptation because our challenge is to infer a right sequence of message modification method candidates, not determining a single optimal modification, which is to be exploited repeatedly for earlier success. Furthermore, the proposed scheme shows a big time complexity which may degrade user experiences in a real world. To overcome these limitations, we propose a new scheme that leverages a protocol knowledge base for identifying message adaptation method candidates and provides a replacement of Exp3, a heuristic algorithm discovering a right adaptation sequence efficiently.

4.2. Architecture overview

As shown in Fig. 1, the proposed scheme establishes two interacting parties: an interaction requester (i.e. user agent) and a target IoT device (i.e. service agent). An interaction requester

Table 1
Primitive message fields and their functionalities.

Functionality	Primitive message fields
Protocol basic Info.	Version Info, Protocol Name
Protocol behavior	Return Code, Op-Code, Control Flag & Rsvd
Session Management	Communication Session Management, Request-Reply Matching, Keep Alive & Expiry
Content Parsing	Message Length, Encoding, Language Code, Query Count, Answer Count, Language Tag, Language Tag Length
Security	Encryption
Service search	Service Type, Service Version, Target Address

brings a legacy protocol and tries to interact with a target IoT device running an updated version of the protocol. The knowledge-based adaptive probing framework resides in the interaction requester and it is composed of four main building blocks: Knowledge Base (KB), Adaptation Controller (AC), Probe Message Composer (PMC), and Probe Message Transceiver (PMT). When an interaction requester starts, KB loads protocol knowledge such as protocol message structures and their updates (Section 4.3). Then, AC leverages that knowledge to initialize an adaptation probability tree which generates a list of possible adaptation method candidates and their relative importance scores (Section 4.4). Based on the scores, AC builds an adaptation sequence planning tree which computes the most probable adaptation sequence whenever asked (Section 4.5). AC queries the most probable adaptation sequence to KB and asks PMC to compose an adapted request message which encodes the built sequence (Section 4.6). Then PMT transmits a message composed by PMC to a target IoT device. When receiving a reply message, an interaction monitor thread reports the result to AC. This routine is called a probing procedure and repeats until a non-error reply message is successfully returned or there is no more adaptation method candidates to try.

4.3. Protocol knowledge base

To understand the relationships among the main functionalities, requirement changes and corresponding message-level changes of a given protocol, it is necessary to investigate various protocol message structures and their statistics. For this, we conducted a thorough analysis on famous application-layer protocols in IoT domain using protocol specification documents and existing protocol surveys [23–25]. The analysis includes well-known service discovery protocols such as SLP, DNS-SD/mDNS, and SSDP/UPnP, and IoT data exchange protocols such as MQTT, CoAP, AMQP, and SIP. Table 1 shows nineteen primitive message fields and their classifications in terms of functionalities. The result includes a collection of fields that are necessary to make each protocol message get parsed successfully and achieve its intended task.

Based on the analysis, we can construct extra knowledge which helps to infer how the protocol would be updated. For example, if we deduce how a message field has been updated, the same update pattern and underlying philosophy can be leveraged to infer future updates of other fields because many practices on protocol messages are performed with the common rationale in protocol engineering. Furthermore, if values for a certain field are unified and not appeared in other recent protocols anymore, the corresponding outdated field is likely to be removed in the next version. Such knowledge can add a bias to make a better inference than a random guess on possible message updates and reach a meaningful interaction more quickly.

For well-defined knowledge base construction, we first design protocol message ontology (Fig. 2) describing the relationships across requirements, functionalities, and corresponding message structures. The protocol message ontology consists of six core classes: *Protocol, ProtocolRequirement, Functionality, ProtocolMessage, ProtocolMessageField*, and *Update*. These core classes are related to each other as follows: In principle, a protocol is designed with respect to a list of requirements and each requirement is related to a certain functionality such as searching a service, describing protocol behaviors, etc. Then, a functionality is realized by multiple fields in a protocol message where each field may contain a numeric or text value. As time passes, message fields are updated by field-specific requirement changes. To represent these rationales, we design predicates such as {*Protocol - hasRequirement - ProtocolRequirement*}, {*ProtocolRequirement - relatedTo - Functionality*}, {*ProtoolMessage - hasField - ProtocolMessageField*}, etc. as shown in Fig. 2. In addition, extra classes (e.g. *RequirementChange*, *UpdatePattern*, etc.) and data properties (i.e. green circles labeled as *'Data'*) are added to describe relationships between individual objects of the defined classes.

Once the protocol information in the same domain is written in the ontology and the set of ontology statements are stored in KB, AC analyzes following statistics to produce the extra knowledge:

- Previous and potential updates (e.g. field length change (L), field deletion (D), new field addition (A), vocabulary change (V), and numeric value change (C)) can be recognized by comparing protocols of different versions. This comparison provides insights into high-level requirement changes and dominant update patterns.
- Relative probability of each field type's appearance can be approximated by counting its appearance frequency. This value implies how common or how important each field is in a target protocol domain. If a certain field type is important, it should appear frequently across various protocol messages.
- Distributions of bit assignments for each field type can be computed. This information can be leveraged later to determine the length of to-be-adapted message fields.

Table 2 shows an analysis result of service request messages in SDP domain, which is to be used for the aforementioned statistics computation. In the table, header field values are denoted by 'h.x' and body field values are denoted by 'b.x', where x indicates the number of bits assigned to the corresponding field. A letter 'v' means the field has a variable length. The field types not present in an SDP message are denoted by a letter '-'. Values in the table cells are created as ontology individuals or data properties and stored in the knowledge base as ontology statements. For example, the knowledge base contains triples such as {*ProtocolBasicInfo - type - Functionality*}, {*Version - hasType - VersionInfo*}, {*Version - encodes - ProtocolBasicInfo*}, etc. After filling up the table entries and converting them to ontology, field appearance probabilities and bit assignment distribution are computed and stored in KB. Table 3 shows the identified update instances from the message structure analysis and their motivations. Since it is yet difficult for a computer agent to infer exactly why message fields are updated without human help, we manually categorize the requirement changes for each update instance. The same analysis is conducted through IoT data exchange protocols and separate knowledge bases are constructed to different protocol domains (i.e. SDP and IoT data exchange protocols). This separation is to prevent different domain's properties which are irrelevant to the current target protocol updates from distracting an efficient and accurate inference.

4.4. Adaptation probability tree

The next step is to build an adaptation probability tree based on the protocol knowledge base built in the step above. The adaptation probability tree is a directed graph and computes the relative

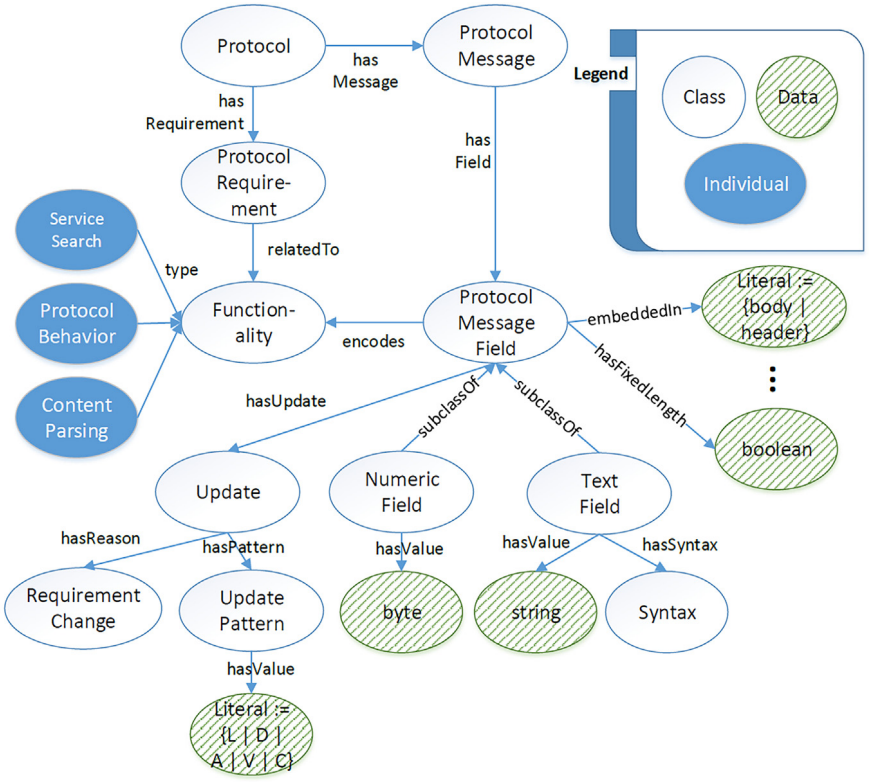


Fig. 2. Protocol message ontology.

Table 2
An analysis of service discovery protocol (SDP) service request message structures.

Functionality	Message field type	# of bits assigned to each field					
		SLPv1	SLPv2	SSDP/UPnPv1	SSDP/UPnPv2	DNS-SD/mDNS	CoAP
Service search	Service type	b.v	b.v	h.v	h.v	b.v	b.v
	Target address	b.32	b.32	h.32	h.32	b.32	b.32
Protocol basic Info.	Version Info.	h.8	h.8	h.v	h.v	–	h.2
Protocol behavior	Return code	–	–	–	–	h.4	h.8
	Message type (op-code)	h.8	h.8	h.v	h.v	h.5	h.8
	Control Flag&Rsvd	h.16	h.40	h.v	h.v	h.7	h.2
Session Management	Comms. Session Mgmt.	h.16	h.16	–	–	h.16	h.16
Content Parsing	Message length	h.16	h.24	h.v	h.v	–	h.4
	Encoding	h.8	–	h.v	h.v	–	–
	Lang. Code	h.16	–	–	–	–	–
	Lang. Tag	–	h.v	–	–	–	–
	Lang. Tag Length	–	h.16	–	–	–	–
	Query count	–	–	–	–	h.16	–
	Answer count	–	–	–	–	h.16	–

Table 3
An analysis on service discovery protocol (SDP) version updates.

Prob.	Update pattern	Requirement change	Updated field	Functionality
0.154	Field length change (L)	Content length change	Message length	Content parsing
		Control option addition	Next Ext Offset	Protocol behavior
0.385	Field deletion (D)	Encoding integration	Encoding	Content parsing
		Language support change	M (Control flag)	Protocol behavior
		Security requirement change	U (Control flag)	Protocol behavior
		Security requirement change	A (Control flag)	Protocol behavior
		Language support change	Language Code	Content parsing
0.231	New field addition (A)	Control option addition	R (Control field)	Protocol behavior
		Language support change	Language Tag	Content parsing
		Language support change	Language Tag Length	Content parsing
0.154	Numeric field value change (C)	Version update	Version	Protocol Basic Info.
		Version update	Version	Protocol Basic Info.
0.077	Vocabulary change (V)	Control option addition	Function type	Protocol behavior

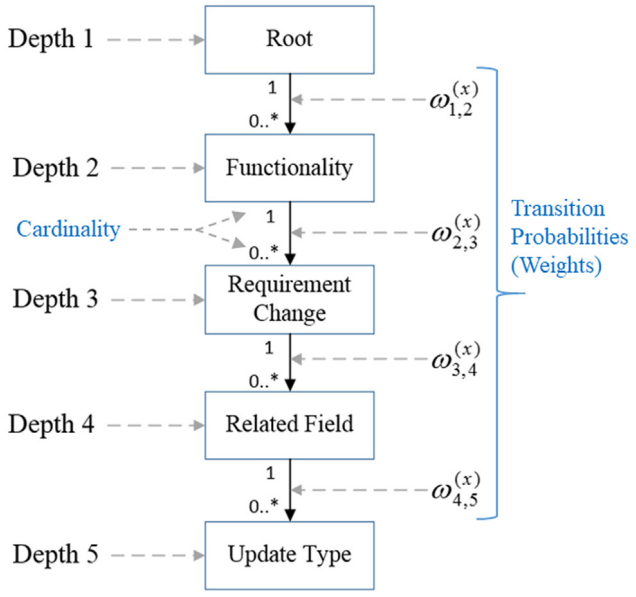


Fig. 3. Adaptation probability tree structure.

Table 4
Semantics of adaptation probability tree nodes.

Depth	Description
1	A starting point of the tree traversal.
2	Identified protocol functionalities to which requirement changes may rise.
3	Possible requirement changes derived from the protocol update history and domain-specific trend (e.g. standardization).
4	Relevant message field types which can be adapted by the selected high-level requirement changes.
5	How to adapt the selected field. One of Addition (A), Deletion (D), Length Change (L), Vocabulary change (V), Numeric Field Value Change (C)

probability of possible adaptation method candidates by means of the weight values assigned to its edges. Fig. 3 shows a tree template and Table 4 describes the semantics of each depth in the tree. A path from a root (depth-1) node to a leaf (depth-5) node represents a message field adaptation method candidate. Let us take as an example tree for SLPv1 in Fig. 4. To allow a longer description (depth-3, requirement change) about message contents (depth-2, functionality), more bits can be assigned (depth-5, adaptation type) to the 'message length' field (depth-4, related field). Onto all edges, numeric values are assigned as transition probabilities toward connected child nodes. What it differs from a conventional transition graph is that a sum of weight values on outbound edges from a node is not necessarily one. This is because the probability of a message field update does not raise or suppress that of the other fields.

The adaptation probability tree contains all possible adaptations in a target protocol domain, not in a specific protocol instance. Therefore, the tree may contain adaptations that may not be applied to a legacy protocol on a user device. For example, if a message field named 'Character Encoding' is present in a service requester's message, 'Addition' (depth-5) of a new 'Character Encoding' field (depth-4) will produce a duplicate field. On the other hand, if an adaptation method candidate suggests to delete a field absent in the legacy protocol message, such adaptation cannot be applied. Therefore, the initial tree should be pruned to filter out the invalid candidates and decrease the search space size.

Once the final list of adaptation method candidates is determined, their possibilities are computed to try more probable ones first. It is done by a function of assigned weight values:

$$p(x) = f(\omega_{1,2}^{(x)}, \omega_{2,3}^{(x)}, \omega_{3,4}^{(x)}, \omega_{4,5}^{(x)}), \tag{1}$$

where x is an adaptation method candidate or a leaf node of depth-5, $p(x)$ is its probability, and $\omega_{i,j}^{(x)}$ is a weight value assigned to every edge on the path from a root note to x. Two indices, i and j, indicate the depths of the start node and the destination node comprising such an edge, respectively. The performance of the candidate detection depends on that of weight value assignment and the definition of the function f, which are computed as follows:

1. $\omega_{1,2}^{(x)}$: This weight value implies the probability of a protocol functionality update. The more frequently the message fields corresponding to a certain functionality get updated in diverse ways, the more probable the functionality is to be updated. Therefore, we compute $\omega_{1,2}^{(x)}$ as a combination of two measures, *update frequency* and *diversity*, as shown in Eq. (2):

$$\omega_{1,2}^{(x)} = \frac{NormUpdateFreq(f_i) + NormDiv(f_i)}{2}, \tag{2}$$

where f_i implies the functionality on the path to leaf node x. Update frequency, $NormUpdateFreq(f_i)$, measures how many times message fields corresponding to each functionality have been updated as well as how likely such updates will occur. Therefore, we leverage the update history of protocol message fields for its computation:

$$NormUpdateFreq(f_i) = \frac{|UpdateHistory(f_i)|}{\sum_{f_i \in F} |UpdateHistory(f_i)|}, \tag{3}$$

where $UpdateHistory(f_i)$ is a set of update instances of message fields corresponding to a given functionality, f_i, and F is a set of all functionalities (Table 3). Diversity, $NormDiv(f_i)$, is a measure of how diversely message fields corresponding to a target functionality is realized in protocol messages. The less diversely message fields corresponding to a certain functionality are realized, the less probable they are to be updated. If all message fields corresponding to a functionality are realized in an identical manner, we determine a solid consensus is made and the fields are not likely to be updated. We compute diversity as shown in Eq. (4):

$$NormDiv(f_i) = \frac{Div(f_i)}{MaxDiv(F)}, \tag{4}$$

where $MaxDiv(F)$ implies the maximum $Div(f_i)$ value among all functionalities in F. By using normalized values for the update frequency and diversity measures, we can scale the measures and $\omega_{1,2}^{(x)}$ value to a range between zero to one. Considering that Shannon's Entropy [26] has been used as a representative measure of diversity across multiple domains, we leverage it for $Div(f_i)$ computation as follows:

$$Div(f_i) = \frac{\sum_{j \in J} H(f_i, j)}{|J|} \tag{5}$$

$$H(f_i, j) = -\sum_{k \in K} p_{i,j}^k \log_2 p_{i,j}^k \tag{6}$$

$$p_{i,j}^k = \frac{freq(k, V_{i,j})}{|V_{i,j}|}, \tag{7}$$

where J implies a set of the corresponding message field types to f_i, K implies a set of bit assignment types to each

field, $V_{i,j}$ implies a set of bit assignment instances for functionality f_i and message field type j, and $freq(k, V_{i,j})$ implies the total appearance count of value k in $V_{i,j}$. In Eq. (5), $Div(f_i)$ is computed as an average entropy, $H(f_i, j)$, of all message field types of a target functionality, f_i. Each entropy value is calculated by the well known entropy Eq. (6) where the probability of a bit assignment type k, $p^k_{i,j}$, is computed as shown in Eq. (7). For example, when f_i is 'Session Management' and all protocols in Table 2 are taken into account, $J = \{$'Comms. Session Mgmt.'$\}$, $K = \{0, 16\}$, $V_{i,j} = \{16, 16, 0, 0, 16, 16\}$, and $NormDiv(F_i) = 0.4787$.

2. $\omega^{(x)}_{2,3}$: This weight value implies how probable each requirement change is made for its connected parent node, high-level functionality. For its computation, we add two measures: evidence and possibility. The evidence implies how often a requirement change has been made based on the protocol update history. For example, when a functionality 'Content Parsing' is given, three types of requirement changes are related to the functionality in Table 3 and each type's appearance count can be leveraged for the evidence value. The possibility implies, regardless of the update history, how many adaptations (depth-5 nodes) are possible under the current requirement changes. When this addition is done for all depth-3 nodes dangling to a functionality (depth-2), the weight value, $\omega^{(x)}_{2,3}$, is computed as:

$$\omega^{(x)}_{2,3} = \frac{evi_{p,c'} + pos_{p,c'}}{\sum_{c \in C_p}(evi_{p,c} + pos_{p,c})}, \tag{8}$$

where p and c' are the functionality and the requirement change on the path from the root to x, $evi_{p,c}$ and $pos_{p,c}$ are evidence and possibility of requirement change c for functionality p, and C_p is the set of dangling requirement changes for functionality p. Based on Eq. (8), a sum of weights on outgoing edges from a functionality p is one. Hence, a weight value on an outgoing edge implies a relative probability to other outgoing edges, which implies a relative probability of a requirement change (c') to other requirement changes (i.e. elements in $C_p - \{c'\}$).

3. $\omega^{(x)}_{3,4}$: Depth-4 and depth-5 nodes are semantically not detachable. For example, we may be interested in the probability of an 'addition of the query count field'. Therefore, we assign a weight value 1 to every edge from depth-3 to depth-4 nodes and use a single weight value, $\omega^{(x)}_{4,5}$, for a path from depth-3 to depth-5 nodes.
4. $\omega^{(x)}_{4,5}$: This weight value implies how probable each dangling adaptation type (e.g. A, D, L, V, C) is. For its computation, the portion of each message field update pattern in the whole update history is used. However, an addition or deletion of a field should be determined accordingly to its existence probability. For example, the 'version information' field is shown in most SDP messages, but DNS-SD/mDNS protocol messages do not have this field in its message (Table 2). Then, it is probable to add the version information field in upcoming updates. To reflect this, a field's addition probability is multiplied by its existence probability.

To discover a depth-5 adaptation method candidate, all connected higher-level nodes should be realized simultaneously. Therefore, to compute the final probability, $p(x)$, we multiply all relevant weight values:

$$p(x) = \prod_{i=1}^{4} \omega^{(x)}_{i,i+1} \tag{9}$$

It is possible that different requirement changes derive the same fields to be adapted in the same pattern. For example, we have two redundant candidates, vocabulary change of 'Control flag & Rsvd', in Fig. 4. One is derived from a security requirement change, but the other is derived from a non-security control option change. In such a case, they need to be integrated into one in order to prevent themselves from being applied to a final message many times. It is the simplest way to average their weight values and pass it to the next step, sequence planning tree construction.

Fig. 4 shows an example adaptation probability tree which is constructed based on the SDP knowledge base in Tables 2 and 3. On each transition edge, a computed weigh value is assigned. In overall, 25 adaptation method candidates are identified, 10 candidates are pruned, and two candidates (Control Flag's Vocabulary Change) are integrated for SLPv1 agent.

4.5. Adaptation sequence planning tree

Once the probabilities of adaptation method candidates are calculated, a right sequence of valid ones should be found. However, even though the pruning step decreases the number of candidates, the number of possible sequences is still too big to try one by one at runtime. Therefore, we leverage a game tree with heuristics [27,28] to decrease the tree's depth and width. We build a game tree whose nodes are adaptation method candidates and leverage the protocol knowledge as heuristics to decrease a search space. This tree is called *an adaptation sequence planning tree* and it returns a sequence with the highest probability for a meaningful interaction.

Fig. 5 shows the structure of an adaptation sequence planning tree. A tree is composed of $N + M$ layers of adaptation method candidates which may include valid and invalid ones. Candidates with lower depth have a higher probability of being valid, and in the same layer, the leftmost candidate has the highest probability. To compose an sequence, we leverage a depth-first tree search algorithm. As the algorithm traverses candidates one by one, a selected candidate is added to an adaptation sequence. When N candidates are added to the sequence so far, M more candidates can be added further. Then the number of adaptation sequences that the sequence planning tree can produce is less than a permutation of M. The final sequence is returned to the probe message transceiver.

Building a full game tree for sequence traversal can take too much memory space. Considering many IoT devices have a small footprint, we need to keep memory usage low. Therefore, rather than building a full tree, we incrementally add tree nodes which need to be traversed but not currently present in the tree. In addition, to prevent allocated memory from running over the capacity of a user's device, we adopt a memoization to free the memory space allocated to tree nodes which will never be visited again. For example, if all subsequences of a tree node are found invalid, that tree node is marked as dead and all memory space allocated to its subsequences can be freed.

To reduce the search space, we adopt heuristics for pruning tree nodes in two-fold: depth and width. To decrease the depth, we bound the adaptation sequence length. Since valid and invalid adaptations are mixed up in the tree, it is not likely that all listed adaptations are performed. Therefore, we determine the most probable length of the adaptation sequence and restrict the tree traversal by that length. This length bound reduces the search space exponentially. To determine the sequence length, we use its distribution over all protocol update history in the knowledge base per domain. To decrease the width, we adopt the high-level knowledge and avoid meaningless adaptations. For example, if a deletion of a field is added previously, we ignore later adaptations on that field.

Fig. 6 shows the adaptation sequence planning tree corresponding to the adaptation probability tree in Fig. 4. We highlight the right adaptation sequence and depict the tried adaptation sequences before it.

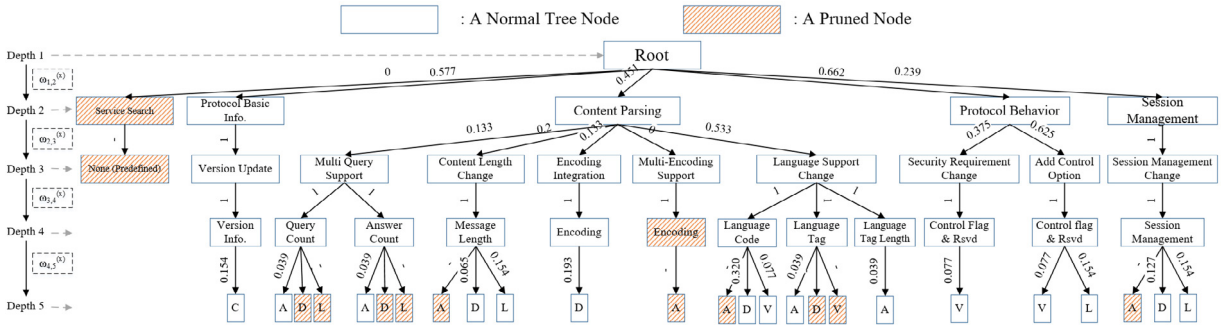


Fig. 4. A pruned adaptation probability tree for SLPv1 user agent.

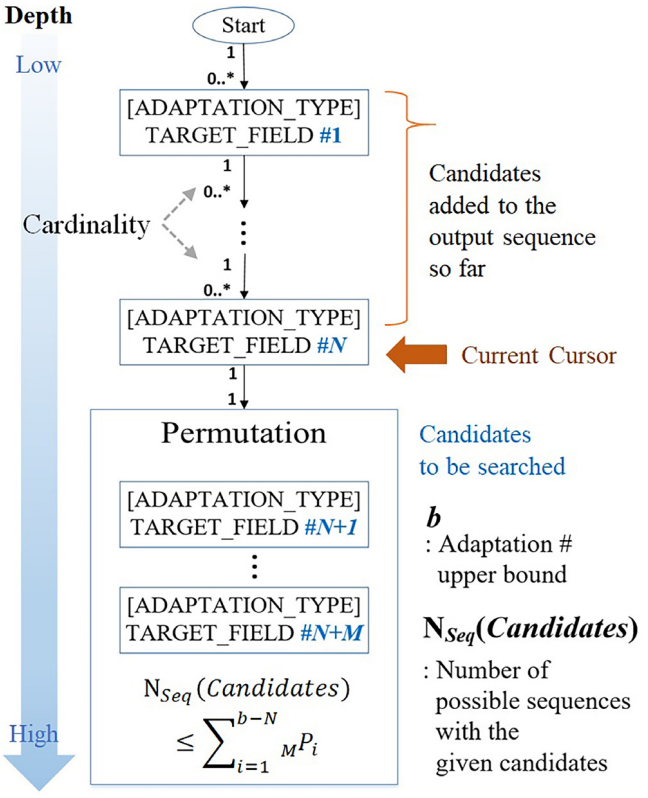


Fig. 5. Adaptation sequence planning tree structure.

4.6. Bit-level message adaptation and probing

When a message-level adaptation sequence is returned, it is passed to the probe message composer (PMC). PMC determines a bit-level adaptation option corresponding to a given message-level candidate based on the message structure analysis and the field length update history in KB (Fig. 7). Once the most probable options are computed, their combination is applied to the original protocol message. Bit-level adaptations for each adaptation type are performed as follows.

When adding a new field ([A]), it would be perfect if we could encode the right semantics of the newly added field even in bit-level so that it is compatible with a new protocol. However, this job may cause too much time complexity, $O(2^n)$ where n is the number of assigned bits. Rather, we replace our goal of adding a field with adjusting the starting point of the next field and the total size of a message. Only when a well-established field is to be added, SeM2Bit tries widely-used standard values first (e.g. "en" for "Language Code" field). Even if right semantics is not delivered, we define it as a meaningful interaction if the structure and the necessary information can be understand by the recipient. Therefore, we add bits with value 0 as much as the adaptation option suggests. For deletion ([D]), we delete a target field and pull the next fields as many bits as the deleted field occupied to fill up the empty space.

When changing a field's vocabulary ([V]), due to the same reason with the field addition, we cannot achieve perfect adaptation which is semantically compatible with a target protocol. Furthermore, a vocabulary change does not require an addition of extra bits. Therefore, we do not perform any bit-level adaptations in this type for the moment and induce an error-prone reply message. As a future work, we plan to add a component which checks an online repository which stores the standard vocabularies such as IANA's database and vendor-specific vocabularies and downloads them if any update is available. A field length change ([L]) can be either incremental or decremental. From the protocol field update history, we can find a pattern that multiple 0 bits are added to an extended field space for padding. For an incremental change, we adopt the same practice and add 0-valued bits as much as suggested. For a decremental change, considering that the least important bits are usually right most fields, we delete the last fields as much as suggested. A numeric value change ([C]) is mostly incremental and applied to version-related fields. For this candidate, the original numeric value changes as the knowledge base suggests.

After a series of bit-level adaptations, the adapted message is returned to PMT and transmitted.

5. Evaluation

For evaluation, we conduct two case studies with protocol update scenarios in different domains. The first case study is designed with service discovery protocols (SDPs) since they are heavily used in IoT environments. In this scenario, a user device uses SLPv1 and encounters a smart device using SLPv2. Since SLPv2 message structure is different from that of SLPv1, the smart device with SLPv2 cannot be discovered. For a meaningful interaction, the SLPv1 user agent keeps adapting its service request message until a reply message is returned. For the second case study, IoT data exchange protocols presented in [23] are leveraged. In this scenario, a user device uses MQTT version 3.1.1 and the corresponding broker uses MQTT version 5. Among many protocol messages, we focus on a CONNECT message because it is the first message that MQTT agents should exchange for further interactions.

5.1. Implementation and experiment setup

We implement SeM2Bit and protocol agents in JAVA on a raspberry pi 3 model B with 1.2 GHz processor and 1 GB RAM, considering its popularity in various IoT applications. We exploit an open

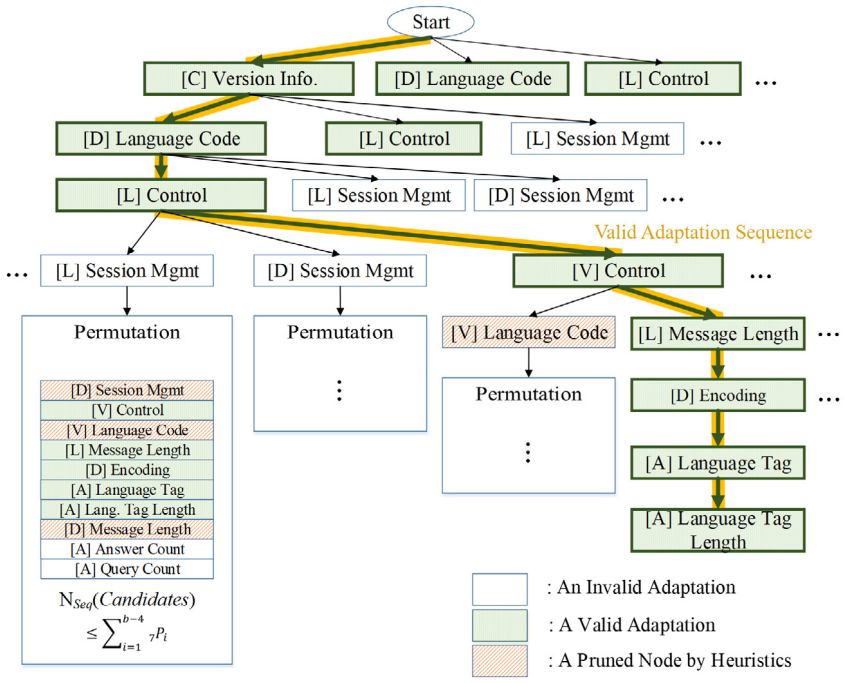


Fig. 6. An adaptation sequence detection for SLPv1.

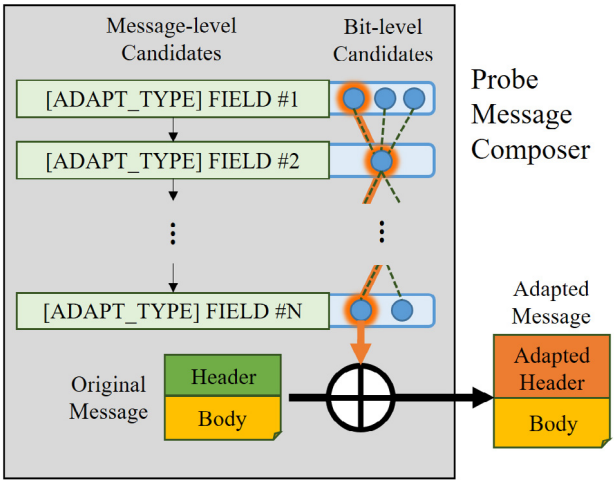


Fig. 7. Bit-level message adaptation in PMC.

source project, jSLP [29] which conforms to SLPv2 and implement SLPv1 as a variation of the current jSLP. To focus on two communicating agents and exclude other parties, SLP agents are running in a P2P mode where Directory Agent (DA) is not involved. For MQTT agent and broker implementations, we leverage two open source projects, Moquette MQTT broker [30] and Paho MQTT client [31]. However, MQTT version 5 specification is recently published and no stable release conforming to the new specification is available. Therefore, we modify Moquette MQTT broker and its underlying Netty framework [32] so that they can handle a CONNECT message conforming to the version 5 specification. Table 5 shows a summary of the implementation and experiment setups.

5.2. Evaluation metrics

To evaluate the accuracy of message inference, we measure precision and recall rate on decisions that SeM2Bit makes to adapt the protocol message for a meaningful interaction. Decisions here include not only applied adaptation methods but also 'still' actions each of which leaves a related field unchanged. We also measure semantic loss which is derived from message adaptations. A meaningful interaction may imply that several fields could be semantically incompatible. Together with accuracy, this metric can give an insight on how SeM2Bit's syntax inference affects corresponding semantics and how much of semantic loss is affordable for a meaningful interaction. We compute this measure as a ratio of the number of missing fields and/or fields which are parsed without an error but whose semantics are different from specification to the total number of expected fields in the updated protocol message. Suppose that an updated protocol message is composed of 10 fields and a meaningful interaction is performed. If 8 fields are parsed but one of them is not semantically compatible, semantic loss in this example is 0.3.

On the other hand, it is also important to verify that the proposed scheme is applicable to real world IoT scenarios where many participants are resource constrained devices and interaction opportunities are not permanent. For that purpose, we analyze the required iteration counts until a meaningful interaction and measure the time taken for the proposed message syntax inference. If inference is successfully done but it takes too much time, the inferred message syntax would not be leveraged. Therefore, inference should be performed quickly enough to hold a user's

Table 5
Implementation and experiment setups.

Setup	Description
Programming language	Java
Target machine	Raspberry pi 3 model B (1.2 GHz processor, 1 GB RAM)
SLP implementation	jSLP [29] library
MQTT implementation	Moquette MQTT broker [30] and Paho MQTT client [31] libraries
Performance profiling	JProfiler [33]

Table 6
Experiment results in SLP case study.

AdaptSeq Length	Iteration count (#)	Precision	Recall	Semantic loss
4	164	0.625	0.5	0.2857
5	1 182	0.667	0.6	0.2857
6	214	0.778	0.7	0.1429
7	326	0.8	0.8	0.1429
8	3,285	0.6	0.6	0.2857
9	3,511	0.818	0.9	0.2857
10	20,457	0.636	0.7	0.2857
11	7,368	0.667	0.8	0.2857
Average	4563.4	0.699	0.7	0.25

attention. We also measure the memory consumption to see what kinds of IoT devices can afford the proposed scheme.

5.3. SLP case study: Iteration count and accuracy

SLP case study results show that SeM2Bit enables meaningful interactions despite 0.7 of adaptation accuracy and 0.25 of semantic loss in average. Table 6 shows the numbers of iterations until a meaningful interaction is made for different adaptation sequence lengths, and corresponding accuracy and semantic loss. SeM2Bit could not find a valid adaptation sequence when the sequence length is less than 4 or larger than 11. Therefore, we investigate meaningful interactions where the length is between 4 and 11 to analyze how the syntax errors and semantic losses are concealed in those cases and SeM2Bit helps to efficiently infer an updated protocol message.

When the length is 4, SeM2Bit shows the best performance in terms of iteration count while the worst performance in the other metrics. Fig. 8 explains how SeM2Bit could enable a meaningful interaction so quickly even with the least accuracy by comparing three headers of the original SLPv1 message, its adapted version, and the expected message by SLPv2 agent. The first three fields of the SLPv1 message header are composed as expected by the SLPv2 agent. However, the extended length of 'Control' flag is 2-bytes shorter than expected, and those 2 bytes are assigned to 'Language Code' field which was supposed to be deleted in SLPv2. While these two mistakes lower the precision, they do not cause any semantic loss in the SLPv2 agent side. This is because the SLPv2 agent parses only the first one byte of control flags and ignores following four bytes which are to be reserved for extensions. 'Char Encoding' field in the adapted header is parsed as an 'XID' field in the SLPv2 agent side, which lower the precision and may cause an error. However, in a simple SLP request-reply routine, inconsistent XIDs do not interrupt the interaction while we can regard it as a semantic loss. Lastly, since 'Language Tag' field contains its length information by itself, 2-bytes of 'Language Tag Length' field is ignored in the SLPv2 agent side. At a result, even though the accuracy is low (0.625 of precision and 0.5 of recall) and 28.57% of information has been lost, the SLPv2 agent could successfully parse necessary fields from the request message and return the information about the requested service.

When the length is 9, SeM2Bit shows the best performance in terms of accuracy but a drastic increase of the iteration count. Fig. 9 shows how this result comes out. SeM2Bit performs most of the valid adaptation methods and it results in a high recall rate, 0.9. However the length of 'Control & Flags' field is incorrectly inferred and the following header fields are shifted forwards all together, which can cause an error in the SLPv2 agent side. Fortunately, for the same reason with when the adaptation sequence length is 4, the fields between the byte indices 6 and 14 are ignored or not used by the SLPv2 agent. This is why wrong adaptation methods performed by SeM2Bit are concealed and the SLPv2 agent successfully reply to a service request message. On the other hand,

Table 7
Correlation coefficients between variables and evaluation metrics.

	Iteration count (#)	Precision	Recall	Semantic loss
AdaptSeq Length	0.677	0.020	0.668	0.252
Iteration count	–	−0.344	0.157	0.385
Precision	–	–	0.743	−0.646
Recall	–	–	–	−0.236

Table 8
Experiment results in MQTT case study.

AdaptSeq Length	Iteration count (#)	Precision	Recall	Semantic loss
2	5	1	1	0
3	4	0.909	1	0
4	3	0.818	1	0
5	2	0.727	1	0
6	7	0.727	1	0
7	31	0.636	1	0
8	121	0.545	1	0
9	361	0.455	1	0

the increase of iteration count is due to the increased adaptation sequence length. The longer an adaptation sequence length is, the more sequences can be composed because it is a simple permutation. In addition, when the length is long, it takes more iterations to recover after putting an invalid candidate into output sequences.

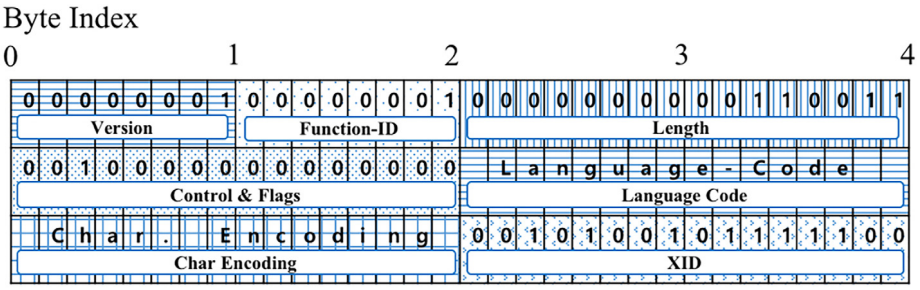
When the length is 7, we conclude that SeM2Bit performs best with moderate iteration count, near-optimal accuracy, and the least semantic loss. Fig. 10 shows the adapted message header when the length is 7 and obviously it looks very close to that of an expected SLPv2 message. In this case, even XID field is positioned precisely, which enables more complicated interactions which require consistent session IDs.

Table 7 shows correlation coefficients between column items in Table 6 and it implies statistical findings including the correlations described above. First, a strong positive correlation is found between adaptation sequence length and iteration count, which is explained in the length-9 case. Another strong positive correlation is found between adaptation sequence length and recall rate. This is because the number of right adaptation methods to enable a meaningful interaction is 10. Obviously, considering that SeM2Bit finds right adaptation sequences whose lengths vary from 4 to 11, as the number of adaptation sequences gets higher (closer to 10), we can get the higher recall rate. Lastly, strong negative correlation is found between precision and semantic loss. This implies that SeM2Bit needs to perform more precise adaptation methods to prevent semantic loss. Unfortunately, we could not find any direct correlations between precision and the other items except recall rate. Since recall rate is not a variable we can manipulate, we need more case studies and experiments to find out how to minimize semantic loss, which we leave for a future work.

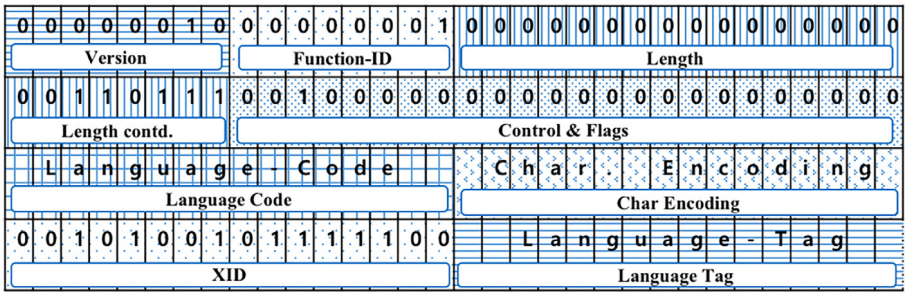
As a conclusion, considering that we have 2.37E+11 possible sequences from 14 adaptation method candidates, exploiting high-level semantics is effective to narrows down the search space and detects valid adaptation methods efficiently. Especially when the sequence length is 4, 6, or 7, much less than a thousand iterations are required, which is drastic improvements compared to a random guess.

5.4. MQTT case study: Iteration count and accuracy

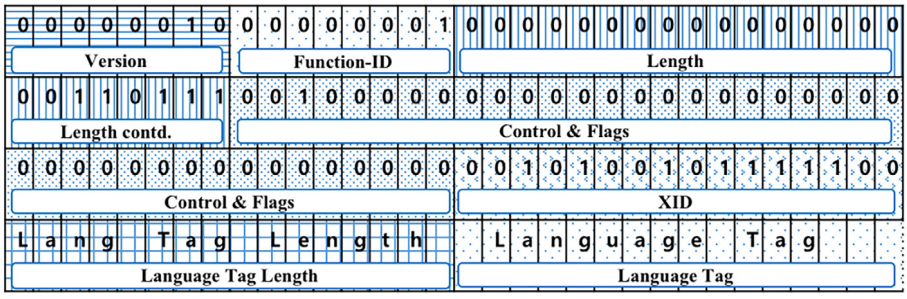
Compared to the SLP case, MQTT update scenario is much simpler. Adaptation probability tree returns 11 candidates, and out of them, the first and the sixth candidates compose the valid adaptation sequence; (1) changing the numeric value of 'Version info', and (2) adding 'Offset' field to include the 'Property Length' value. SeM2Bit cannot make a meaningful adaptation when the sequence



(a) Original SLPv1 Service Request Message Header



(b) Adapted SLPv1 Service Request Message Header (SeqLen = 4)



(c) Expected SLPv2 Service Request Message Header

- Performed Adaptation Methods :
  - <[C] Version> <[L] Control> <[L] Length> <[A] Language Tag>
- Accuracy Computation : ('[-]' means 'leave it unchanged')
  - Valid Actions = {[C] Version, [-] Function-ID, [L] Length (+8), [-] XID, [A] Language Tag}
  - Wrong Actions = {[L] Control (+8), [-] Language Code, [-] Char Encoding}
  - Missing Adaptation Methods = {[L] Control (+24), [V] Control, [D] Encoding, [D] Language Code, [A] Language Tag Length}
  - Precision = 5/(5+3) = 0.625, Recall = 5/(5+5) = 0.5
- Semantic Loss in SLPv2 Header field : 28.57%
  - Valid = {Version, Function-ID, Length, Control & Flags, Language Tag}
  - Lost Semantics = {XID, Language Tag Length}

Fig. 8. Adapted SLPv1 message header and evaluation metric computations (Sequence length = 4).

length is one because the valid sequence is composed of two candidates. In addition, SeM2Bit fails when the length is larger than nine as well. This is because the returned 11 candidates contain 'Addition of language code' (with the fifth highest probability) and 'Addition of character encoding' (with the least probability), which interrupts a meaningful interaction if applied. When the length is larger than nine, at least one of these 'must-avoid' adaptation method is definitely applied.

From the experiment results in Table 8, we observe that precision decreases continuously, while recall rate and semantic loss never change from 1 and 0, respectively. This is because the number of applied adaptation methods increases but we have only two valid candidates. The more adaptation methods are applied, the more wrong actions are performed by SeM2Bit. On the other hand, recall rate is never degraded because all performed meaningful interactions are attributed to the aforementioned two valid adaptation methods. In addition, since other invalid adaptation

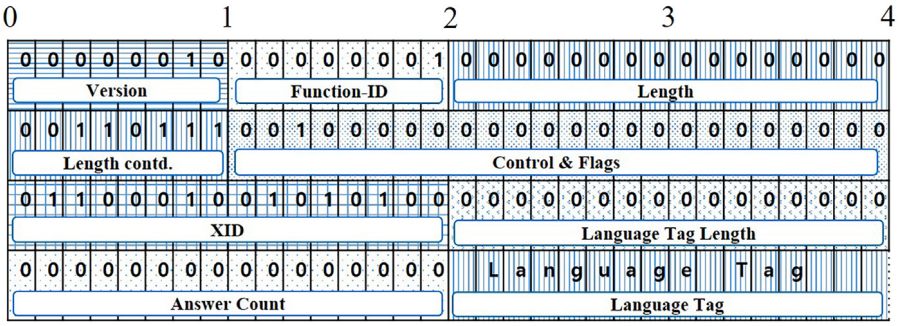


Fig. 9. Adapted SLPv1 message header (Seq. Length = 9).

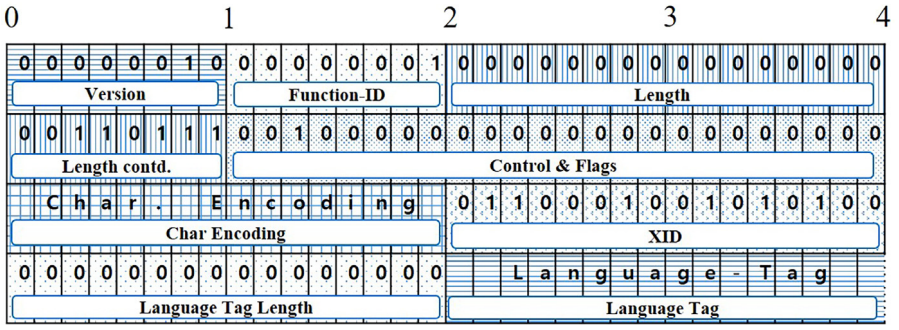


Fig. 10. Adapted SLPv1 message header (Seq. Length = 7).

methods applied to the CONNECT message are all of 'Vocabulary change' and SeM2Bit can do nothing with that update pattern for the moment, no semantic loss is caused.

We also find out that the interaction count is the smallest when the sequence length is 5, and afterwards, it drastically increases as the length becomes larger. Basically, the necessary and sufficient condition for a meaningful interaction is application of two valid candidates and exclusion of 'Addition of language code' and 'Addition of character encoding'. Considering the order of the valid and the 'must-avoid' candidates, given that the valid ones are chosen, the probability that at least one of the 'must-avoid' candidates is chosen becomes higher as the sequence length increases beyond 5. This explains the growing tendency of the interaction count.

Like SLP case study, the result shows that exploiting high-level semantics is effective to narrow down the possibilities and efficiently draw a meaningful interaction.

5.5. Overhead: Elapsed time and memory usage

The underlying hypothesis of SeM2Bit is that protocol domain knowledge helps to infer the updated protocol message efficiently. To prove the hypothesis, we analyze the most straightforward efficiency measures, elapsed time and memory usage, in SLP and MQTT case studies as shown in Tables 9 and 10, respectively. First, we observe that the elapsed time is proportional to the iteration count given in the previous sections. Considering that the sequence length is a random variable following a Gaussian distribution whose mean value is the average number of entailed field changes per a protocol version update, SeM2Bit is highly probable to choose 6 or 7 in SLP and 3 in MQTT case study as the first sequence length to try. Then, in case of SLP, it takes 5 to 8 s of the elapsed time for message inferencing. According to a well-established response time analysis [34], this much of time is short enough to keep a user's attention focused on the current interaction. Considering that SLP is a passive and non time-critical protocol by nature, keeping a user's attention is enough to discover a service agent with an updated protocol. In case of MQTT, it takes 533 ms when the adaptation sequence length is 3. According to the response time analysis [34], this much of elapsed time is short enough to make users perceive their interactions seamless. Even though MQTT communication is timely decoupled, it often requires near-realtime data exchanges in many IoT applications. From that perspective, SeM2Bit does not harm the performance of MQTT even when different versions of service and user agents interact with each other. Furthermore, according to multiple CPU benchmark scores [35–37], this elapsed time can be shortened when SeM2Bit runs on more powerful IoT devices such as smart phones. Therefore, we conclude that the underlying hypothesis of SeM2Bit works in terms of the elapsed time and usability.

The hypothesis works in terms of memory usage as well. The memory usages including heap (before Garbage Collection) and non-heap spaces are 27.24 MB and 12.92 MB in the worst case, respectively in SLP and MQTT case study. After GC, the memory usages decrease down to 18.13 MB and 10.42 MB. However, SeM2Bit hardly suffers from the worst case due to the knowledge-based sequence length restriction. After all, SeM2Bit causes a small memory overhead which can be afforded even by resource-constrained IoT devices. IoT devices here include those who are deployed to interact with each other, not a tiny sensor node which only publishes data periodically on a predefined network topology.

6. Discussions

6.1. How to minimize semantic loss

In the SLP case study, we found that a meaningful interaction can be made despite of semantic loss, 0.25 in average. Even though a semantic loss does not completely disrupt the meaningful interactions, a question is if it could do when intended interactions happen in a more complex scenario. For example, the semantics of 'XID' was often lost due to incorrect positioning and the loss may affect following interactions (e.g. message retransmission) which

Table 9
SeM2Bit overhead in SLP case.

AdaptSeq Length	Elapsed time (ms)	Heap usage before GC (MB)	Heap usage after GC (MB)	NonHeap usage (MB)
4	4,816	4.98	2.00	6.00
5	25,571	6.05	2.82	6.18
6	5,754	5.82	2.38	6.10
7	7,952	5.96	2.06	6.13
8	69,248	7.26	3.25	6.16
9	73,909	7.39	4.29	6.18
10	422,430	21.01	11.90	6.23
11	152,276	13.06	5.8	6.20

Table 10
SeM2Bit overhead in MQTT case.

AdaptSeq Length	Elapsed time (ms)	Heap usage before GC (MB)	Heap usage after GC (MB)	NonHeap usage (MB)
2	538	5.49	2.65	6.01
3	533	5.28	2.65	6.00
4	524	5.00	2.64	6.01
5	517	4.82	2.63	5.99
6	603	5.74	2.44	6.03
7	1,955	6.30	2.60	6.21
8	6,363	6.62	2.94	6.3
9	18,727	7.60	4.02	6.4

require the precise session ID. To make a legacy protocol message 'perfectly compatible' to its updated version, the current design goal should be more advanced so that we can minimize semantic loss in critical fields which may affect following interactions. In addition, the presented key building blocks should be redesigned accordingly.

Above all, the adaptation sequence planning tree needs to be improved to engage KB in the sequence computation. Since all semantic losses found in the case study are caused by incorrect positioning, more precise message syntax inference is necessary for minimizing semantic losses. It requires the sequence planning tree to fetch the critical fields from KB. Then, the depth-first tree traversal algorithm should be replaced with more sophisticated algorithm that (1) preserves the critical fields' positions and (2) locates well-fitted candidates before and after the critical fields, (3) while reflecting the order of the candidates' probabilities.

In addition, 'Control & Flags' field is not counted as a semantic loss. However, if an application requires SLPv2's extended options, its service agent should parse the reserved bits in 'Control & Flags' field. Then, semantic loss increases and a meaningful interaction can be interrupted. The same argument is applied to the current SeM2Bit's no-semantics-inference policy for 'Vocabulary change' update pattern. To overcome this issue, SeM2Bit should be able to search for external knowledge about the message fields (e.g. RFCs, wikipedia, etc.) and download necessary information to infer future updates on field semantics. For that purpose, we need to devise an additional semantic query engine by adopting existing state-of-the-art mechanisms such as ontology-based [38] or text mining-based [39] web knowledge extraction.

6.2. Complexity analysis and a potential extension towards a formal method

Since SeM2Bit is based on an iterative heuristic algorithm, its complexity is dependent on the protocol knowledge base. Despite the positive results from two case studies, it is meaningful to formally analyze the proposed scheme's complexity because it can help to estimate its performance when applied to another protocol. The complexity analysis can be conducted upon two main procedures, adaptation method candidate extraction in the initialization phase and next adaptation sequence computation in the inference phase. In the former phase, since candidate extraction is performed only once during the initialization and never performed again, its time complexity analysis is pointless. However, its space complexity matters because a tree structure is stored to encode a protocol knowledge. We restrict the number of dangling nodes in depth 2 and 5 to specific constants and those in depth 3 and 4 hardly exceed a small number (e.g. maximum 5 in SLP case). Therefore, we conclude that the adaptation method candidate extraction has a space complexity of $O(1)$.

In the next adaptation sequence computation phase, space complexity is $O(1)$ by the virtue of the memoization. However, time complexity is more critical than space complexity in this phase. When n candidates are passed by the adaptation probability tree and the adaptation sequence length is b, the theoretical time complexity is $O(n^b)$ when the right sequence is detected after all sequences of length b in the sequence planning tree are tried. However, its expected number of iteration counts in real scenarios should be much smaller than that because SeM2Bit applies adaptation sequences in the order of their probabilities, which roughly form a long-tailed distribution. To approximate how much the *higher-probability-first* policy can lower the time complexity, a distribution of the sequences' probabilities should be specified. However, it is difficult because we do not have enough protocol update history samples and no prior work has statistically analyzed SDP requirement changes. To compute even a naive approximation of the probability distribution with our limited number of samples, several approximation techniques can be leveraged. For example, we formulate the probabilities as a power function by means of a regression analysis. Then, we deduce an exponential distribution whose probability density function shows the similar property with the power function. By using its mean value, it is expected that the worst case of the SLP case study can be lowered down to 19.28% and 28.15% when the adaptation sequence length, b, is six and seven, respectively. Six or seven is selected as the first adaptation sequence length according to the update history as explained in Section 5.5. In addition, considering that message adaptations are occasionally successful even with a semantic loss, the expected iteration count can be lowered further.

When the target is not a formal computation model, it is challenging to generalize a tight bound of the adaptation counts as Juba does with EXP3 algorithm. A possible extension for this issue is integrating a formal model such as Angluin's $L*$ algorithm [18] to SeM2Bit. Basically, protocol syntax and behavior can be regarded as a language and its grammar. Considering that $L*$ algorithm is an automata learning technique dedicated for learning a language, it can be a powerful and generic solution for learning not only

protocol behavior but also protocol message syntax. Key challenges for doing that are defining an effective equivalent query to verify the counterpart protocol message (i.e. language) and designing an algorithm which efficiently selects minimum number of membership queries. Our current intuition is that the protocol domain knowledge can play a critical role to find out the optimal membership query. Such domain-specific optimization in automata learning is proven to be effective [40] and we expect its application to turn SeM2Bit into a solid computational model with bounded adaptations.

6.3. Contributions and concerns toward future generation computing systems

Recently, many architectures [1–3,41–43] have been proposed with regard to Edge/Fog Computing, Vehicular Networks, Cyber–Physical Systems (CPS), and 5G. In such architectures, beyond the traditional client–server model, various computing entities including small sensors/actuators and autonomous vehicles actively participate in user-centric service provisioning. For example, Vehicular Content Centric Networks (VCCN) [2] leverages intermittent connectivities among vehicles and infrastructures for V2X communications and Opportunistic Edge Computing (OEC) [41] leverages spontaneous interaction opportunities between edge computing entities. However, due to the dynamic nature of the participating entities, it is highly challenging to satisfy key requirements in 5G era [1]; reliable communications and low latency.

Specifically, runtime protocol incompatibility is one of the critical issues entailed by the dynamic nature, which can degrade both requirements. For example, in an edge environment, virtualized resources and functions are dynamically orchestrated according to user or application requirements [3,42]. However, when communication protocols of collocated nodes in the same edge cloud are not compatible, even if the nodes' functionalities can logically compose an orchestration to fulfill a given service request, underlying physical communication between the nodes cannot be realized and the service request should be forwarded to other edges or a remote cloud server. Eventually, service latency inevitably increases and service providers cannot rely on the edge communication.

The proposed scheme can enhance the runtime protocol compatibility between edge nodes and help to relieve such challenge. If the proposed inference scheme is applied to various protocol domains and incompatible protocols become more interactable at runtime, edge nodes can successfully exchange messages with a higher probability. Then, a service request is more likely to be fulfilled by collocated edge nodes, which makes it unnecessary to interact with the other edges or a remote server. In addition, if the result of the proposed inference mechanism can be transferred to other edge nodes or base stations (e.g. edge controller and 5G base station) as a protocol update knowledge, we can prevent multiple edge nodes from performing redundant inferences of the same target protocol by reusing the inferred knowledge hosted by the closest node.

Despite the potential contribution, our proposal may bring a cybersecurity concern in the future computing environments. In edge computing environments, participating edge nodes have certain granted privileges to use and process nearby data [43]. While this privilege is necessary to enable the edge computing, malicious nodes can abuse the privilege to steal private information, overuse other nodes' computing resources without an authority, or break down the overall computing systems. To prevent those malicious behaviors, many participants can keep their interaction/security protocols up-to-date because those with outdated protocols can be naïve targets for malicious nodes. However, if the malicious nodes are equipped with protocol inference techniques such as our proposal and Automatic Protocol Reverse Engineering [13,14], those techniques can be leveraged to break the cybersecurity in edge computing environments. To relieve this security concern while encouraging active interactions between edge nodes, platform or service providers should adopt appropriate framework to identify malicious nodes and adapt access authorities of edge nodes. If necessary, identified malicious nodes which keep trying bad attacks should be excluded eventually. For that purpose, Sohal et al. [43] present a cybersecurity framework which categorizes malicious nodes and leverages their logs to defend future attacks. Evaluating trustworthiness of participant nodes and adapting their authorities accordingly to the trustworthiness [44,45] can be another cybersecurity solution.

7. Conclusion

In this paper, we propose SeM2Bit, an efficient runtime protocol message inference scheme to achieve meaningful interactions with unknown but backward incompatible updates. The proposed scheme infers protocol messages for the updated version of a legacy protocol based on the protocol knowledge without corresponding specifications at runtime. For that purpose, we construct a domain knowledge base including the message structures and update patterns of existing protocols. Then, we devise two tree-based algorithms to identify adaptation method candidates for a legacy message and compute the most probable sequence of them. To improve their efficiency, multiple pruning mechanisms are applied to reduce the depth and width of the trees. For evaluation, we conduct two case studies with SLP and MQTT update scenarios. The results show that SeM2Bit enables meaningful interactions despite a certain level of accuracy degradation and semantic loss with an affordable level of computation and memory overhead.

For future works, we plan to advance SeM2Bit as a faithful solution for the 'universal semantic communication' [17] with which incompatible versions of protocols exchange more complex sequence of messages. This would enable SeM2Bit to achieve not only a meaningful message exchange but also a secure and reliable communication between incompatible versions by means of supporting authentication and retransmission. At the same time, it is an important challenge to guarantee the similar or better performance than the experiment results in Section 5 and make the performance verifiable. We will develop semantic loss minimization which allows meaningful interactions to be achieved even when protocols' extended features are activated. This involves redesigning KB and adaptation sequence computation algorithm and devising an extra semantic query engine as explained in Section 6. In addition, making SeM2Bit's heuristic inference scheme computationally verifiable is another challenge. We can try to model the current scheme as it is, but leveraging an existing formal method such as automata learning is considered to provide a more solid framework to start with.

References

[1] S.A.A. Shah, E. Ahmed, M. Imran, S. Zeadally, 5g for vehicular communications, IEEE Commun. Mag. 56 (1) (2018) 111–117.
[2] A. Wahid, M.A. Shah, F.F. Qureshi, H. Maryam, R. Iqbal, V. Chang, Big data analytics for mitigating broadcast storm in vehicular content centric networks, Future Gener. Comput. Syst. 86 (2018) 1301–1320.
[3] G. Sun, Y. Li, D. Liao, V. Chang, Service function chain orchestration across multiple domains: A full mesh aggregation approach, IEEE Trans. Netw. Serv. Manag. 15 (3) (2018) 1175–1191.
[4] M. Eslamichalandar, K. Barkaoui, H.R. Motahari-Nezhad, Dynamic adapter reconfiguration in the context of business protocol evolution, in: Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on, IEEE, 2013, pp. 301–308.
[5] N.D. Ryan, A.L. Wolf, Using event-based translation to support dynamic protocol evolution, in: Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, 2004, pp. 408–417.

[6] Network Working Group, Service Location Protocol version 1, URL https://tools.ietf.org/html/rfc2165, 1997.
[7] Network Working Group, Service Location Protocol version 2, URL https://tools.ietf.org/html/rfc2608, 1999.
[8] A. Bennaceur, V. Issarny, Automated synthesis of mediators to support component interoperability, IEEE Trans. Softw. Eng. 41 (3) (2015) 221–240.
[9] N. Bencomo, A. Bennaceur, P. Grace, G. Blair, V. Issarny, The role of models@ run. time in supporting on-the-fly interoperability, Computing 95 (3) (2013) 167–190.
[10] T. Pramsohler, S. Schenk, A. Barthels, U. Baumgarten, A layered interface-adaptation architecture for distributed component-based systems, Future Gener. Comput. Syst. 47 (2015) 113–126.
[11] M. Merten, F. Howar, B. Steffen, P. Pellicione, M. Tivoli, Automated inference of models for black box systems based on interface descriptions, in: International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, Springer, 2012, pp. 79–96.
[12] Y. Wang, X. Li, J. Meng, Y. Zhao, Z. Zhang, L. Guo, Biprominer: Automatic mining of binary protocol features, in: Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on, IEEE, 2011, pp. 179–184.
[13] J.-Z. Luo, S.-Z. Yu, Position-based automatic reverse engineering of network protocols, J. Netw. Comput. Appl. 36 (3) (2013) 1070–1077.
[14] J. Caballero, P. Poosankam, C. Kreibich, D. Song, Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering, in: Proceedings of the 16th ACM conference on Computer and communications security, ACM, 2009, pp. 621–634.
[15] M. Liu, C. Jia, L. Liu, Z. Wang, Extracting sent message formats from executables using backward slicing, in: Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on, IEEE, 2013, pp. 377–384.
[16] H. Freudenthal, Lincos, design of a language for cosmic intercourse, part I, Studies in Logic and the foundations of mathematics.
[17] B. Juba, Universal Semantic Communication, Springer Science & Business Media, 2011.
[18] D. Angluin, Learning regular sets from queries and counterexamples, Inf. Comput. 75 (2) (1987) 87–106.
[19] Y.-D. Bromberg, P. Grace, L. Réveillère, Starlink: runtime interoperability between heterogeneous middleware protocols, in: Distributed Computing Systems (ICDCS), 2011 31st International Conference on, IEEE, 2011, pp. 446–455.
[20] R. Agrawal, R. Srikant, et al., Fast algorithms for mining association rules, in: Proc. 20th int. conf. very large data bases, VLDB, Vol. 1215, 1994, pp. 487–499.
[21] B. Juba, Compatibility among diversity foundations, lessons, and directions of semantic communication, in: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on, IEEE, 2013, pp. 458–463.
[22] P. Auer, N. Cesa-Bianchi, Y. Freund, R.E. Schapire, The nonstochastic multi-armed bandit problem, SIAM J. Comput. 32 (1) (2002) 48–77.
[23] A. Rayes, S. Salam, Iot protocol stack: A layered view, in: Internet of Things From Hype to Reality, Springer, 2017, pp. 93–138.
[24] P. Sethi, S.R. Sarangi, Internet of things: architectures, protocols, and applications, J. Electr. Comput. Eng. (2017).
[25] A. Aijaz, A.H. Aghvami, Cognitive machine-to-machine communications for internet-of-things: A protocol stack perspective, IEEE Internet Things J. 2 (2) (2015) 103–112.
[26] C.E. Shannon, A mathematical theory of communication, ACM SIGMOBILE Mobile Comput. Commun. Rev. 5 (1) (2001) 3–55.
[27] M.L. Littman, Reinforcement learning improves behaviour from evaluative feedback, Nature 521 (7553) (2015) 445–451.
[28] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.
[29] J. Rellermeyer, jslp project, java service location protocol (2008).
[30] M. Moquette, Moquette MQTT, URL https://projects.eclipse.org/projects/iot.moquette, 2017.
[31] eclipse paho, eclipse Paho (1.2.0), URL https://www.eclipse.org/paho/, 2017.
[32] Netty project, Netty project (4.1.25), URL http://netty.io/, 2018.
[33] ej technologies, JProfiler, URL https://www.ej-technologies.com/products/jprofiler/overview.html, 2018.
[34] J. Nielsen, Usability Engineering, Elsevier, 1994.
[35] The TOFFEE Project, TrueBench: CPU Benchmarks, URL http://truebench.the-toffee-project.org/index.php?page=home&lang=, 2018.
[36] PassMark Software, PassMark, URL https://www.passmark.com/index.html, 2018.
[37] PRIMATE LABS, Geekbench 4, URL https://www.geekbench.com/, 2018.
[38] H. Alani, S. Kim, D.E. Millard, M.J. Weal, W. Hall, P.H. Lewis, N.R. Shadbolt, Automatic ontology-based knowledge extraction from web documents, IEEE Intell. Syst. 18 (1) (2003) 14–21.
[39] G. Weikum, M. Theobald, From information to knowledge: harvesting entities and relationships from web sources, in: Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, 2010, pp. 65–76.
[40] H. Hungar, O. Niese, B. Steffen, Domain-specific optimization in automata learning, in: International Conference on Computer Aided Verification, Springer, 2003, pp. 315–327.
[41] R. Olaniyan, O. Fadahunsi, M. Maheswaran, M.F. Zhani, Opportunistic Edge Computing: Concepts, Opportunities and Research Challenges, arXiv preprint arXiv:1806.04311.
[42] B.S. Si Young Jang, Yoonhyoung Lee, D. Lee, Application-aware iot Camera virtualization for video analytics edge computing, in: Proceedings of the Third ACM/IEEE Symposium on Edge Computing, ACM, 2018.
[43] A.S. Sohal, R. Sandhu, S.K. Sood, V. Chang, A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments, Comput. Secur. 74 (2018) 340–354.
[44] B. Gwak, H. Son, J. Kang, D. Lee, Iot trust estimation in an unknown place using the opinions of I-sharing friends, in: Trustcom/BigDataSE/ICESS, 2017 IEEE, IEEE, 2017, pp. 602–609.
[45] B. Gwak, J.-H. Cho, D. Lee, H. Son, Taras: Trust-aware role-based access control system in public internet-of-things, in: 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), IEEE, 2018, pp. 74–85.

Heesuk Son received the B.S. degree in School of Computing from Korea Advance Institute of Science and Technology (KAIST), Korea in 2011. He is currently pursuing his Ph.D. degree at the School of Computing department in KAIST. His current research interests include IoT, communication protocol, pervasive computing, smart spaces, data mining, and agent-based intelligent system.

Dongman Lee received the B.S. degree in Computer Engineering from Seoul National University, Korea in 1982, and the M.S. degree and Ph.D. degree in Computer Science from KAIST, Korea in 1984 and 1987, respectively. From 1988 to 1997, he worked as Technical Contributor at Hewlett-Packard. Since 1998, he has been professor of School of Computing at KAIST. He has published over 190 papers in international journals and conferences. He has served as TPC member and chair of numerous international conferences including IEEE COMPSAC, Multimedia, PDCS, PERCOM, PRDC, VSMM, ICAT, etc. and a reviewer of international journals and magazines including ACM TOMCCAP, IEEE TPDS, IEEE Proceedings, IEEE JIE, IEEE TWC, Computer Networks, TOCSJ, JCN, IEEE wireless communication magazine, and IEEE Intelligence magazine. His research interests include pervasive computing, edge computing, social computing, and mobile computing. He is a member of KISS and IEEE, and a senior member of ACM.